Experimenting a Conflict-Driven Clause Learning Algorithm

Gilles Audemard, Laurent Simon

Experimenting Small Changes in Conflict-Driven Clause Learning Algorithms

Gilles Audemard[1] and Laurent Simon[2]

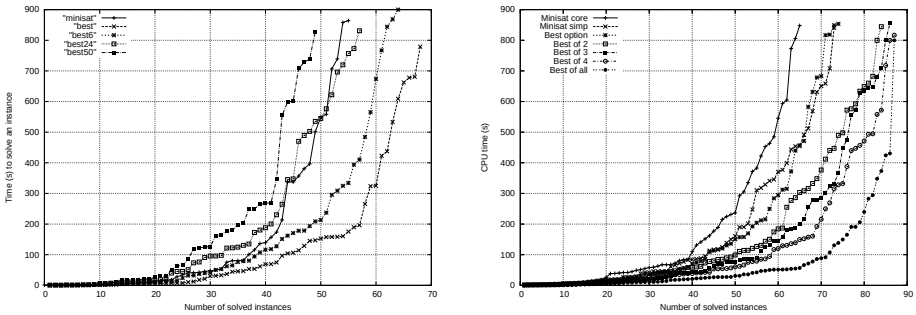
[1] Univ Lille-Nord de France CRIL / CNRS UMR8188,
Lens, F-62307 audemard@cril.fr,
[2] Univ Paris-Sud, LRI / CNRS UMR8623 / INRIA Saclay
Orsay, F-91405 simon@lri.fr

Abstract. Experimentation of new algorithms is the usual companion section of papers dealing with SAT. However, the behavior of those algorithms is so unpredictable that even strong experiments (hundreds of benchmarks, dozen of solvers) can be still misleading. We present here a set of experiments of very small changes of a canonical Conflict Driven Clause Learning (CDCL) solver and show that even very close versions can lead to very different behaviors. In some cases, the best of them could perfectly have been used to convince the reader of the efficiency of a new method for SAT. This observation can be explained by the lack of real experimental studies of CDCL solvers.

1 Introduction

Conflict-Driven Clause Learning algorithms (CDCL) have been one of the major breakthroughs in the practical solving of industrial SAT problems. Since the introduction of ZChaff in 2001 [8], a lot of progresses have been made [3], and solvers can now tackle problems of millions of clauses. All techniques and methods embedded in "modern" solvers are well known: dynamic heuristics [8, 4], learning [9], restarts [6, 1] and lazy data structures [8]. Efficient solvers can nowadays be written from scratch in less than a thousand lines of code.

However, we believe that the underlying mechanisms are still not understood. They result from extensive tests rather than strong experimental studies, where paradigms would be proposed and tested against observations. We believe that new breakthroughs in the next years may only come if we begin to really understand the reasons of solvers performances. A new technique may be good, but can still be thrown away and not published because of a dramatic side effect of a previously unknown behavior of CDCL solvers. It is thus crucial to begin an in-depth study of modern solvers, without trying to improve their performances at first. In this short paper, we try to consider a typical CDCL solver, MINISAT [3], as a physical system that we try to test against well admitted ideas. Our final aim here is more to cast new questions to the community, given some observations of MINISAT performances, rather than proposing a full and tested paradigm of CDCL solvers. As a side effect of our studies, we illustrate how far one may improve MINISAT performances with only a couple lines hack. This last observation may for instance be a standard to know whether or not new solvers bring really new ideas or may result from a side effect of small changes of a canonical CDCL solver, MINISAT.



a. Shuffle Neighboorhood b. One-Parameter Neighboorood

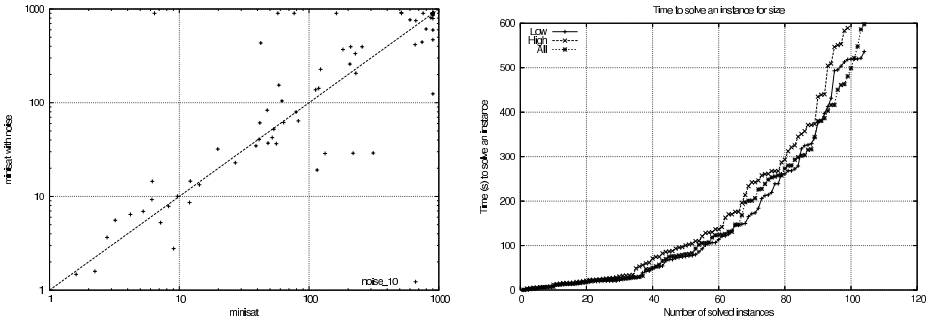
Fig. 1. Studying Shuffling (left) and Parameters (right) (median time on 40 launches)

2 Shuffling Effects: the Lisa Syndrome, revisited

It is well admitted that shuffling instances have a negative effect on industrial benchmarks (see the so-called "Lisa" Syndrom in [7], related to [2]). This observation has motivated, in SAT Contests and SAT Races, to consider only proper benchmarks. Thus, it is admitted that modern solvers explicitly use a heuristic that suppose a non-shuffled instance. Over all explanations, one intuition is that first variables have more chances to be input variables than additional variables introduced in order to avoid combinatorial explosion. During the first phase, MINISAT chooses decision variables in lexicographic order (some solvers choose variables according to their occurrences in the input formula). However, it is not clear how much one may lose by shuffling an instance. If the above explanation holds, and if order of clauses and variables are related to some important structural property between clauses and literals, then one should loose a lot by shuffling an instance before calling MINISAT.

All experiments are done with instances of the SAT Race 2006.Shuffling is done on variables order, clauses order and literals order in each clause (like in [2]).

Figure 1.a shows the traditional performance plot for solvers comparison. It gives the CPU time (in seconds) needed to solve a given number of instances. We can read that MINISAT (without SATELITE preprocessing) is able to solve 55 problems. Curves *"best"* (respectively *"best6"*, *"best24"* and *"best50"*) plot the result of virtual solvers which would have the best CPU time obtained on all 50 shuffled runs (respectively the 5th, 24th and 50th (median) percentile). Thus, a very simple shuffling (50 times) of instances allows to solve 69 instances in less than 900 seconds (in comparison to the 55 instances solved with only one run on the original problem). What is more striking, is that there is 24% of chances to obtain better results by shuffling instance (see "best24" curve). This result is clearly higher than one would have predicted if it was only justified by the topology of the original problem (input/output variables encoding).



a. 10% Noisy Parameters b. Effect on clause deletion by size

Fig. 2. Noisy parameters (left, median on 40 runs) ; Clauses deletion (right, median on 20 runs)

3 Parameters effects

When tuning the solver, a number of parameters have to be set (like the randomness of the heuristics, the number of conflicts before restarts, ...). We study how performances can be enhanced by changing only one of these values in MINISAT. We took 10 different magic values of MINISAT parameters and studied all (1-parameter) neighboors as they were different solvers. For each value, we tried both MINISAT with SATELITE (called *simp*) and without it (called *core*), on all original benchmarks. Between 5 and 8 different values were tested for each of 10 parameters[3], which give us 126 solvers (half with SATELITE).

Figure 1.b gives the results for some *virtual* solvers based on parameter neighborhood. Each *best of N* curve corresponds to the subset of N solvers that give the best results, if the N solvers were ran in parallel on N computers. First observation: using two versions of MINISAT (the best couple of solvers were core with `RESTARTINC`=1 and simp with MINISAT default values) can pay a lot. It seems that keeping a very fast restart policy, but without preprocessing, may pay. This shed a new light on recent works on restart policies. We report the best of 3 solvers, also based on variants of restart policies: MINISAT simp with default values, MINISAT core with `RESTARTINC`=1 and MINISAT core with `RESTARTINC`=1.1.

[3] `VARDECAY` (inverse of the variable activity decay) $\in \{0.5, 0.75, 0.85, 0.90, 0.95, 0.99, 0.999\}$; `VARINC` (init. amount to bump vars) $\in \{1, 2, 5, 10, 50\}$; `RESTARTINC` (factor by which the restart limit is multiplied after restarts) $\in \{1, 1.1, 1.25, 1.5, 1.75, 2, 4, 8\}$; `RESTARTFIRST` (init. restart limit) $\in \{10, 50, 100, 200, 500, 1000, 5000, 10000, 50000\}$; `RANDOMVARFREQ` (frequency with which MINISAT choose a random variable rather than the heuristics based one) $\in \{0, 0.001, 0.002, 0.003, 0.01, 0.05, 0.1, 0.5\}$; `LEARNTSIZEINC` (factor that increases the limit of learnt clauses) $\in \{0.5, 0.8, 1, 1.1, 1.2, 1.5, 2, 4\}$; `LEARNTSIZEFACTOR` (limit for learnt clauses as a factor of the total number of clauses) $\in \{1, 1.5, 2, 3, 4, 5, 8\}$; `CLAINC` (init.amount to bump clauses with) $\in \{1, 2, 5, 10, 50\}$; `CLAUSEDECAY` (inverse of the clause activity decay factor) $\in \{0.5, 0.75, 0.85, 0.90, 0.95, 0.99\}$; `POLARITYMODE` (branching) $\in \{false, true\}$

The second observation is based on the proximity of all best-N curves (except for the best of all, that even though, joins all best-N curves at the end), which means that MINISAT really reaches its limits there. One may cast doubts on the real improvement of CDCL solvers if any brand new solver does not really improve this "hard" limit.

Figure 2.a reports another experiment: we took MINISAT and, each time one of the 10 constants was requested, we added 10% random noise to it. We can see that the "noisy" MINISAT now behaves like another solver. When new methods exhibit similar performance plot w.r.t MINISAT, nothing can be really drawn from it. This can only be due to some hidden noise. Last observation we made: When considering the whole neighborhood, using SATELITE as a preprocessor is not so important. We measured that differences between best of all simp versions and best of all core versions are only by one more bench solved for the first version.

4 Learning large or short clauses?

In order to avoid memory explosion, modern solvers clean out learnt clauses database. Clauses with less activity (the number of times that these clauses were directly, and recently, considered when analyzing the reasons for the conflict) are deleted. However, it is not necessary for CDCL solver completeness to keep learnt clauses until the end. They just have to provide a reason for current asserting literals. This reason, represented as a clause, can be forgotten when it becomes unnecessary. We analyze here the behavior of MINISAT when one forces it to forget some clauses. Our first goal is to know whether some classes of clauses may be removed without degrading MINISAT performances. The second is more important. We believe that improvements of future CDCL solvers are related to highlighting "*important*" learnt clauses (yet another time, in a multi-core context, it would be worth sharing a clause between processes only if it is *important*, see [5] for example).

We conducted this experiment as follows. First, we run MINISAT on shuffled instances (20 times), and store, for each benchmark, the median size of learnt clauses. Then, we run 3 versions of MINISAT. The first one forgets 25 % of learnt clauses of any size. In the second (resp. third), it forgets 50% of clauses of size less than (resp. greater than) the computed median size (for a given benchmark). For each parameter, and each benchmark, we consider the median CPU time over 20 shuffled instances. This experiment should show what is highly believed: the size of learnt clauses matters.

Results are summarized in figure 2.b and, contrary to what is usually believed, it seems that short clauses are not *significantly* more important than large ones. Indeed, removing short, large or any clauses produces approximatively the same results at the end. This was already pointed out in previous, theoretical, works, that shown that some proofs need large clauses, but it is surprising to measure in practice that deleting 50% of short clauses is not so different than deleting 50% of large clauses. We also tried to characterize important clauses with other parameters (number of resolutions step during conflict analysis, minimal resolution depth of clauses), but results are identical, and *important* clauses are very hard to characterize, with a global measure.

5 Conclusion

In [2], it was already proposed to use shuffling techniques to characterize the behavior of solvers, and to begin a real experimental study of them. However, this is not a sufficient framework to really test solvers against hypothesis, as they were physical systems. This work is a first step in this direction. We took a canonical, well known, solver, MINISAT, and built experimental studies in order to validate or invalidate some well admitted ideas. So, what can be drawn from our very simple experiments? First, shuffling instances is not as bad as one may have expected. In 25% of the case, it may pay, which is probably to high to confirm that the locality of variables and the order of clauses in real world problems really matters. It is often argued that shuffling instances is useless and has no meaning at all from a practical point of view. However, if one wants to add good learnt clauses somewhere in a formula, by any preprocessing technique, then it is essential to understand where to add it, and if the order really matters and how. At last, we showed that it is not possible to consider short clauses as globally more important than large clauses, which is highly counter-intuitive and was believed to be false. We also show that, by moving parameters, one may obtain really different solvers.

In the next years, CDCL framework will probably be extended to multi-core architectures, which will increase their complexity and their "unpredictability". If one wants to understand their behavior, a lot of effort has to be made now. Would it be satisfactory to use the multi-core ability of next processors generation only by using different shuffled instances of the same benchmarks? We shown in this paper that a lot of progress has to be done in order to really, deeply, understand why CDCL are so efficient, and what mechanisms are essential. In the quest for efficiency, it is urgent to begin to study them, from a real, deep, experimental perspective.

References

1. A. Biere. Adaptive restart strategies for conflict driven SAT solvers. In *Proceedings of SAT'08*, pages 28–33, 2008.
2. F. Brglez, X.Y. Li, and M. F. Stallmann. On SAT instance classes and a method for reliable performance experiments with SAT solvers. *Annals of Mathematics and Artificial Intelligence*, 43:1–34, 2005.
3. Niklas Een and Niklas Sorensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
4. Eugene Goldberg and Yakov Novikov. BerkMin: A fast and robust SAT-solver. *Discrete Applied Mathematics*, 155(12):1549–1561, 2007.
5. Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: Solver description. In *System description for the SAT-RACE*, 2008.
6. Jinbo Huang. The effect of restarts on the efficiency of clause learning. In *proceedings of IJCAI'07*, pages 2318–2323, 2007.
7. D. Le Berre and L. Simon. Essentials of the SAT'03 competition. *SAT'03*, 2003.
8. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC'01*, pages 530–535, 2001.
9. João P. Marques Silva and Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.